University of California, Los Angeles
Department of Computer Science

CS 174A: Intro to Computer Graphics
Final Project

Team BLR
05 June 2020

CS 174A
Section 1A, Fri 12pm
Prof. Asish Law

Contents

1 Introduction

For our project, we proposed a simple basketball shooting game in which a player has 2 minutes to score as many points as possible by shooting a basketball into a hoop on the other side of the scene. During development we shifted the scoring system to a randomly placed target on the wall to ease the development constraints and create a more challenging game for the player. The game is built entirely using the tiny-graphics.js software library.

Our code can be found here:
https://github.com/intro-graphics/team-project-team-blr/blob/master/examples/basketball.js

2 Team

2.1 Name

Our team name is "Team BLR."

2.2 Members

We have 3 members on our team.

- Brian Le
  - UID: 404-731-460
  - GitHub handle: @brianle20
  - Email: brian.le1678@gmail.com
- Logan Hernandez
  - UID: 205-020-619
  - GitHub handle: @logan-hernandez
  - Email: logankhernandez@gmail.com
- Ricky Ho
  - UID: 804-751-314
  - GitHub handle: @ricky-ho
  - Email: horicky55@yahoo.com

3 How to Play the Game

To play the game, the user controls the ball's movement and trajectory using their mouse cursor. Click and hold to move the ball around and release to throw the ball. The velocity of the ball depends on the speed and direction in which the player drags and releases their mouse. The

goal of the game is to hit as many targets to score as many points as possible within the time limit. When the timer ends, the player can simply continue playing by clicking on the screen again which will automatically restart the timer and reset the score.

4 Game Design

The environment of the game was created using tiny-graphics' shapes which were translated, rotated, and scaled. For example, the ground and the walls are basically squares that were first rotated to the correct orientation, translated, and then scaled. The textures for the objects and shapes are mapped by defining the texture attribute with square image files for the Material using the Textured Phong shader. All of the images used for the textures were found using Google search.

The game is 2 minutes long - for the first 1 minute and 30 seconds, hitting the target scores the player 1 point. The last 30 seconds is a bonus round where the player can score 5 points per target hit. If at any point the score exceeds the high score, the high score is updated. When the timer hits zero, the game stops and the score is reset to zero. The player may restart the game by picking up the ball again.

The ball operates under one of two different types of physical behavior at a time: following the position of the mouse or following the force of gravity. Because of the two distinct behaviors, the ball is either drawn as a shape or a body. When it is drawn as a shape, its location transform matrix is translated in the xy-plane according to the mouse position. After the mouse click is released, the ball is drawn as a body with an initial position of the previous ball and an initial velocity calculated by the mouse velocity. The body then follows the initial vector and the force of gravity defined by the class `Simulation` in the `collisions-demo.js` file. Then the body toggles back to a shape when the player clicks for the next shot.

The target on the wall is a subdivision sphere(4) that is made into a Body for collision detection to work. If a collision is detected between the target and the basketball, then the target is popped and a new target is generated with random x and y coordinates on the wall.

5 Advanced Features

We implemented three advanced features - collision detection, mouse picking, and physics - to determine the mechanics of the ball and the target.

5.1 Collision Detection

The collision detection in our game uses the collision detection written in tiny-graphics in the `collisions-demo.js` file in the Body class. In this class, the function `check_if_colliding( b, collider )` is responsible for detecting collisions. The collisions algorithm utilizes bounding volumes that roughly conform to the drawn shape and detects if any of these volumes intersect. The only bodies in our game are the basketball and the target on the wall, which is essentially a flattened sphere, thus the algorithm only needs to check for collisions between them. Thus in our Basketball Game class function `update_state( dt )`, we need only to call `check_if_colliding(`

`ball, target )` to determine if they have collided or not. If a collision is detected, then we will increment the score for the game and pop the target from its array and create a new target at a random location on the wall.

5.2 Mouse Picking

For mouse picking we added listeners on three mouse events: `mousemove`, `mousedown`, and `mouseup`. When each of these events occur, a function is called which determines the ball's behavior. When the mouse is clicked down the mouse's position is calculated based on its position in the client with the origin at the center of the window. This value is then used to determine how much to translate the ball in the xy-plane. While the mouse is clicked down, if the mouse moves then the transformation matrix that determines the translation is recalculated so the ball follows the mouse. When the mouse click is released, the ball's velocity is calculated and the ball is rendered as a body rather than a shape.

To calculate the velocity, the last ten time instances of the mouse's x-position and y-position are recorded for the average velocity. When a click is released, the mouse velocity is scaled by a constant vector aimed towards the target to create an artificial vector component in the z-direction since the mouse can only move in the xy-plane. This is used as the initial linear vector for the physical body calculations for the ball.

5.3 Physics

The framework for our physics came from the new version of tiny-graphics' `collisions-demo.js` file. In order to implement the physics, we needed to import the classes Body and Simulation and make our Basketball Game class a child of Simulation. This allowed us to create the basketball as a Body. The physics of the ball is implemented in the Basketball Game class function `update_state( dt )`.

The basketball is given an initial linear velocity in the y-direction which is updated with the game's delta time (dt) multiplied by -1.8. Normally, the multiplied value would have been -9.8 to emulate Earth's gravitational force, but we had to scale down to create a more realistic and smooth looking effect. The linear velocity of the ball is dampened whenever the ball impacts the floor or the walls. The angular velocity is given a small arbitrary value to mimic the natural rotation of the ball in the physical world which also decays as the ball slows down.

6 Contributions

6.1 Brian Le

Gameplay, Text, Physics

- Physics
- Combine mouse picking with gravity physics
- Gameplay + Text

6.2 Logan Hernandez

Mouse Picking, Physics

- Mouse picking and physics
- Mouse velocity calculations
- Launch vector calibration

6.3 Ricky Ho

Environment, Textures, Target on Wall, Collision Detection

- Static Environment
- New Tiny-graphics Version
- Collision Detection
- Create Random Target

Please refer to the repository to see the full commit history.